بسمه تعالی

فرشید حسین‌زاده ۹۸۲۱۲۰۳ **مینی پروژه دوم**

سوال اول.

ابتدا دیتاست را دانلود کرده و بر روی گوگل کولب بارگذاری می‌کنیم:

```
!pip install --upgrade --no-cache-dir gdown
!gdown 13H9qGbsruLMG_CYNJZbzV2-azcP1Q09f
```

سپس کتابخانه‌های مورد نیاز را اضافه کرده و همچنین دیتاست را می‌خوانیم:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

df = pd.read_csv('/content/Perceptron.csv')
df
```

در ادامه ستون ویژگی‌ها را به X و ستون کلاس را به y نسبت می‌دهیم:

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
print(X)
print(y)
```

بخش اول:

حال داده‌ها را به دو بخش آموزش و تست تقسیم می‌کنیم:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((320, 2), (80, 2), (320,), (80,))
```

در این بخش از random state = 3 استفاده شده است. (دو رقم آخر شماره دانشجویی)

حال یک نورون را برای این مجموعه آموزش می‌دهیم:

```
perceptron = Perceptron()
perceptron.fit(X_train, y_train)
```

بخش دوم:

در این بخش دقت را بر روی مجموعه تست بدست می‌آوریم. برای این کار، ابتدا یک تابع دقت تعریف می‌کنیم که مقدار آستانه را در آن برابر با صفر قرار می‌دهیم:

```
[19] def accuracy(y, y_hat, threshold = 0):
       y_hat = np.where(y_hat < threshold, -1, 1)
       acc = np.sum(y == y_hat) / len(y)
       return acc
```

حال با استفاده از این تابع و متود perceptron.predict ، میزان دقت را محاسبه و چاپ می‌کنیم:

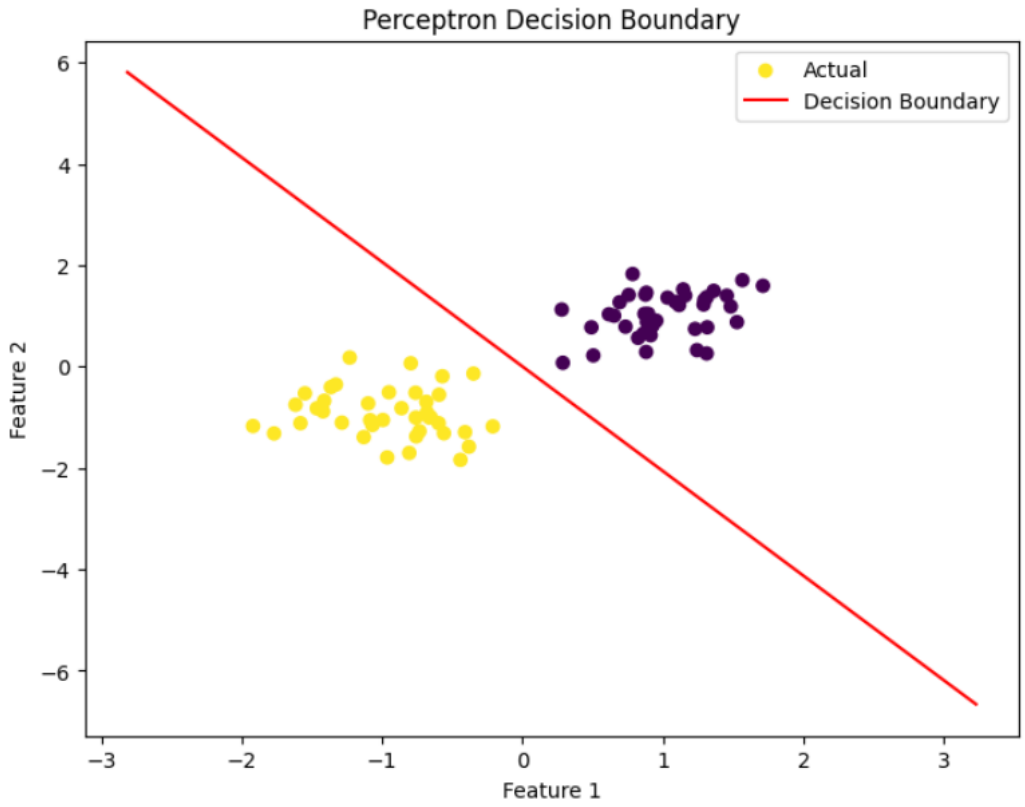
```
y_pred = perceptron.predict(X_test)

acc = accuracy(y_test, y_pred)
print(f"Accuracy on test set: {acc * 100:.2f}%")
```

```
Accuracy on test set: 100.00%
```

برای نمایش خط جداکننده برای داده‌های تست و همچنین نمایش جداگانه کلاس‌ها، با استفاده از روش زیر اقدام می‌کنیم:

```
if X_train.shape[1] == 2:
    w = perceptron.coef_[0]
    b = perceptron.intercept_

    x0 = np.linspace(np.min(X_train.iloc[:, 0]) - 1, np.max(X_train.iloc[:, 0]) + 1, 100)
    x1 = -(w[0] * x0 + b) / w[1]

    plt.figure(figsize=(8, 6))
    plt.scatter(X_test.iloc[:, 0], X_test.iloc[:, 1], c=y_test, cmap='viridis', label='Actual')
    plt.plot(x0, x1, color='red', label='Decision Boundary')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Perceptron Decision Boundary')
    plt.legend()
    plt.show()
```

که در این قسمت ابتدا وزن‌ها و بایاس استخراج شده و سپس مقادیر X و Y برای خط تصمیم‌گیری به تعداد ۱۰۰ عدد تولید می‌شود. سپس نمودار را با مقادیر واقعی داده‌های تست ایجاد می‌کنیم. نتیجه به شکل زیر خواهد بود:



بخش سوم:

در این بخش با تغییر آستانه در بازه ۱- تا ۱ تغییری در نتیجه حاصل نمی‌شود اما با خارج شدن مقدار آستانه از این بازه، پنجاه درصد دقت را از دست می‌دهیم که با توجه به اینکه مقدار تارگت‌ها در دو سر این بازه است، منطقی است.

```
def accuracy_new(y, y_hat, threshold = 0.5):
  y_hat = np.where(y_hat < threshold, -1, 1)
  acc = np.sum(y == y_hat) / len(y)
  return acc
```

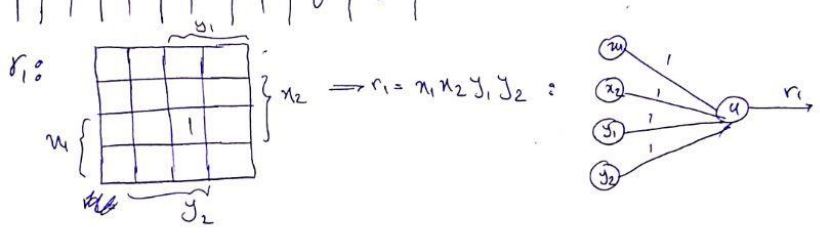
```
y_pred = perceptron.predict(X_test)

acc = accuracy_new(y_test, y_pred)
print(f"Accuracy on test set: {acc * 100:.2f}%")
```

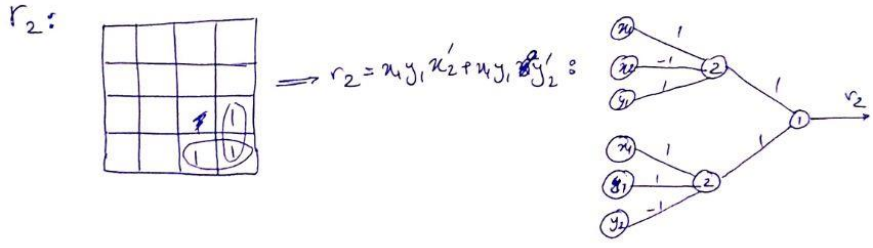
```
Accuracy on test set: 100.00%
```

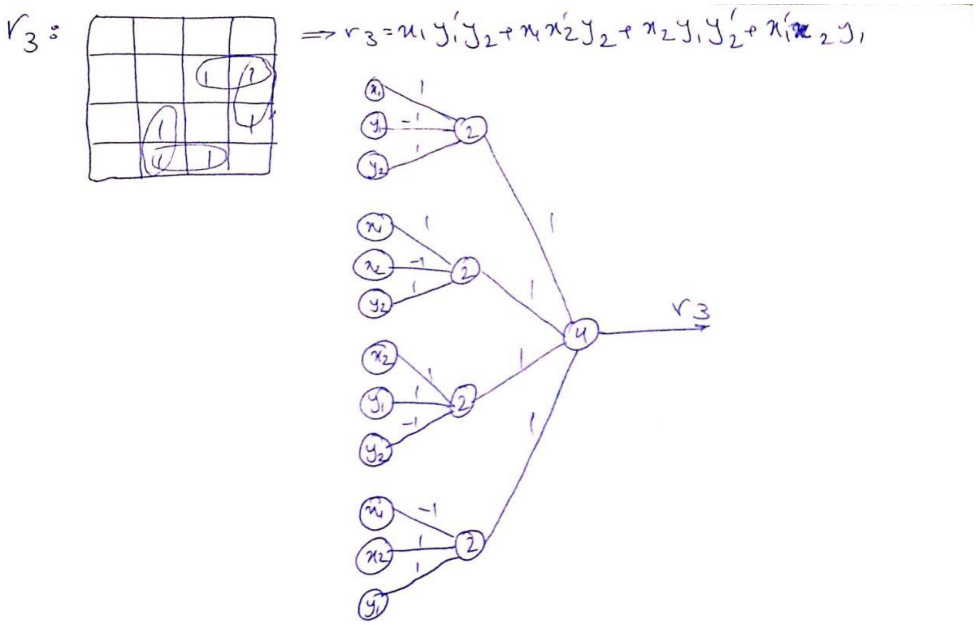
سوال دوم.

برای این سوال نیاز به رسم جدول درستی و سپس جداول کارنو برای هر خروجی داریم. ورودی‌ها را به ترتیب x1,x2,y1,y2 در نظر می‌گیریم که دو بیت ورودی اول و دو بیت ورودی دوم هستند و خروجی‌ها هم به ترتیب r1,r2,r3,r4 هستند. با استفاده از جداول کارنو، شبکه خروجی به همراه وزن ها و مقادیر آستانه را حساب می‌کنیم. در تصاویر زیر این محاسبات را می‌بینیم:

x_1	x_2	y_1	y_2	r_1	r_2	r_3	r_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1







بخش دوم:

برای پیاده‌سازی این شبکه با زبان پایتون، کلاسی به نام McCulloch_Pitts تعریف می‌کنیم. این کلاس دو آرگومان وزن و مقدار آستانه را می‌پذیرد. در داخل این کلاس، تابعی به نام مدل تعریف شده است که بررسی می‌کند آیا حاصل ضرب ورودی در ماتریس وزن بزرگ‌تر از مقدار آستانه است یا خیر. اگر بزرگ‌تر بود ۱ و اگر کوچک‌تر بود، ۰ برمی‌گرداند.

```python
import pandas as pd
import numpy as np
import itertools

class McCulloch_Pitts():

  def __init__(self, weights, threshold):
    self.weights = weights
    self.threshold = threshold

  def model(self, x):
    if self.weights @ x >= self.threshold:
      return 1
    else:
      return 0
```

سپس یک تابع ضرب کننده تعریف می‌کنیم که برای هر خروجی نورون‌های مجزا ایجاد می‌کند تا خروجی مورد نظر را بدست آورد.

```python
def multiplier(input):
  neur1 = McCulloch_Pitts([1, 1, 1, 1], 4)
  r1 = neur1.model(np.array([input[0], input[1], input[2], input[3]]))

  neru21 = McCulloch_Pitts([1, -1, 1], 2)
  r21 = neru21.model(np.array([input[0], input[1], input[2]]))
  neru22 = McCulloch_Pitts([1, 1, -1], 2)
  r22 = neru22.model(np.array([input[0], input[2], input[3]]))
  neru2 = McCulloch_Pitts([1, 1], 1)
  r2 = neru2.model(np.array([r21, r22]))

  neru31 = McCulloch_Pitts([1, -1, 1], 2)
  r31 = neru31.model(np.array([input[0], input[2], input[3]]))
  neru32 = McCulloch_Pitts([1, -1, 1], 2)
  r32 = neru32.model(np.array([input[0], input[1], input[3]]))
  neru33 = McCulloch_Pitts([1, 1, -1], 2)
  r33 = neru33.model(np.array([input[1], input[2], input[3]]))
  neru34 = McCulloch_Pitts([-1, 1, 1], 2)
  r34 = neru34.model(np.array([input[0], input[1], input[2]]))
  neru3 = McCulloch_Pitts([1, 1, 1, 1], 1)
  r3 = neru3.model(np.array([r31, r32, r33, r34]))

  neru4 = McCulloch_Pitts([1, 1], 2)
  r4 = neru4.model(np.array([input[1], input[3]]))

  y = [r1, r2, r3, r4]

  return y
```

برای داشتن نتیجه:

```
a = [0, 1, 0, 1]
X = list(itertools.product(a, a, a, a))
input = []
for i in range(0, len(X)):
  n = input.count(X[i])
  if n == 0:
    input.append(X[i])

for i in range(0, len(input)):
  r = multiplier(input[i])
  print(str(input[i]) + ': result of the multiplication ' + str(r))
```

نتیجه:

```
(0, 0, 0, 0): result of the multiplication [0, 0, 0, 0]
(0, 0, 0, 1): result of the multiplication [0, 0, 0, 0]
(0, 0, 1, 0): result of the multiplication [0, 0, 0, 0]
(0, 0, 1, 1): result of the multiplication [0, 0, 0, 0]
(0, 1, 0, 0): result of the multiplication [0, 0, 0, 0]
(0, 1, 0, 1): result of the multiplication [0, 0, 0, 1]
(0, 1, 1, 0): result of the multiplication [0, 0, 1, 0]
(0, 1, 1, 1): result of the multiplication [0, 0, 1, 1]
(1, 0, 0, 0): result of the multiplication [0, 0, 0, 0]
(1, 0, 0, 1): result of the multiplication [0, 0, 1, 0]
(1, 0, 1, 0): result of the multiplication [0, 1, 0, 0]
(1, 0, 1, 1): result of the multiplication [0, 1, 1, 0]
(1, 1, 0, 0): result of the multiplication [0, 0, 0, 0]
(1, 1, 0, 1): result of the multiplication [0, 0, 1, 1]
(1, 1, 1, 0): result of the multiplication [0, 1, 1, 0]
(1, 1, 1, 1): result of the multiplication [1, 0, 0, 1]
```